HY16F 系列

IDE 软件最佳化使用说明书

目录

注意：

1、本说明书中的内容，随着产品的改进，有可能不经过预告而更改。请客户及时到本公司网站下载更新 http://www.hycontek.com

2、本规格书中的图形、应用电路等，因第三方工业所有权引发的问题，本公司不承担其责任。

3、本产品在单独应用的情况下，本公司保证它的性能、典型应用和功能符合说明书中的条件。当使用在客户的产品或设备中，以上条件我们不作保证，建议客户做充分的评估和测试。

4、请注意输入电压、输出电压、负载电流的使用条件，使IC内的功耗不超过封装的容许功耗。对于客户在超出说明书中规定额定值使用产品，即使是瞬间的使用，由此所造成的损失，本公司不承担任何责任。

5、本产品虽内置防静电保护电路，但请不要施加超过保护电路性能的过大静电。

6、本规格书中的产品，未经书面许可，不可使用在要求高可靠性的电路中。例如健康医疗器械、防灾器械、车辆器械、车载器械及航空器械等对人体产生影响的器械或装置，不得作为其部件使用。

7、本公司一直致力于提高产品的质量和可靠度，但所有的半导体产品都有一定的失效概率，这些失效概率可能会导致一些人身事故、火灾事故等。当设计产品时，请充分留意冗余设计并采用安全指标，这样可以避免事故的发生。

8、本规格书中内容，未经本公司许可，严禁用于其他目的之转载或复制。

1 简介

本文主要介绍 Andesight 的最佳化设置与其他相关事项

2 最佳化介绍与设置

设置步骤：打开 project 后，选择菜单栏中的 project—>properties 后弹出下图，在 settings—>optimization 中可选择需要的优化级别

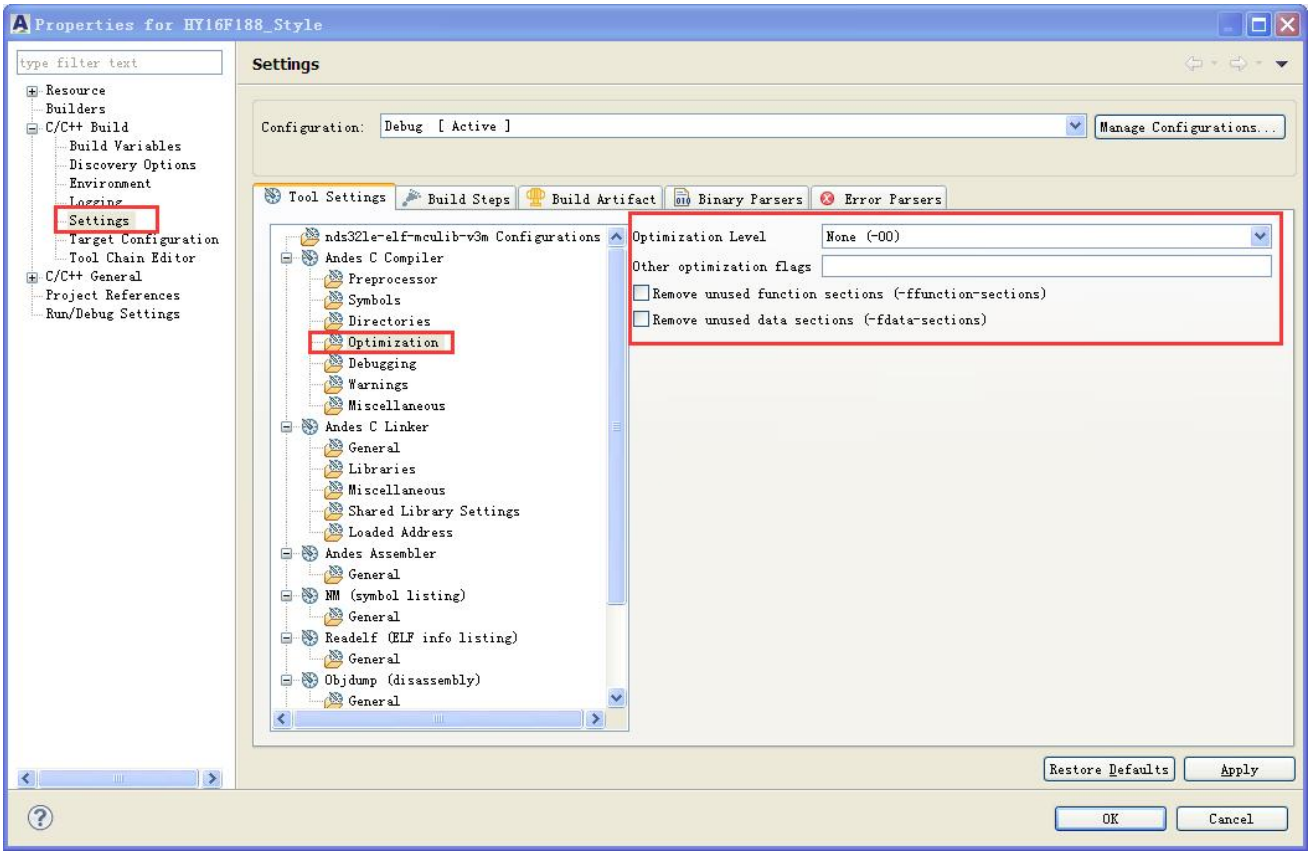


图 1

Optimization level 的含义可参考如下信息：

Andesight 将代码的占用空间和执行速度优化级别各分为 3 个级别。

一般情况下程序的代码空间的最佳化会选择-Os3 和勾选 remove unused section；执行速度最佳化会选择-O3 和加上-funroll-loops（可参考图 2）。

(Optimize speed)-O/O1：GCC 将执行减少代码尺寸和执行时间的优化，对于那些会严重影响编译时间的优化选项，这个级别的优化并不会执行。

(Optimize speed more)-O2：-O 的进阶，在这一级别 GCC 将会提供所有支援的优化，但这其中并不包括以空间换时间的优化，例如编译器不会使用循环展开和函数内联。和-O 相比，该选项进一步加快了编译时间和生成代码的性能。

(Optimize speed most)-O3 ：除了-O2 提供的优化选项外，还指定了-finline-functions，-funswitch-loops 和-fgcse-afer-reload 等选项，-O3 优化级别提升生成的可执行档的速度，但也可能增加它的大小。在有些情况下也可能会使得程序运行减慢。

(Optimize size most)-Os3 (-Os)：这个选项是用来优化代码尺寸，-Os 打开了所有-O2 级别中不会显著增长代码尺寸的优化选项，同时-Os 还会执行更加优化程序空间占用的选项。因此，有时-Os 可能会影响程序性能，可以选择其他较低的优化级别代替-Os

(Optimize size more)-Os2：相比-Os 减少使用-mifc 等选项，只打开部分的代码优化选项.

(Optimize size)-Os1：代码空间优化深度较小，对代码的影响也较小，只打开部分代码优化选项

图 2 中的 Remove unused function sections(-ffunction-sections)和 Remove unused data sections (-fdata-sections)其功能是将没有使用到的函数和数据删除，以减少代码占用空间，如有需要请勾选。在不使用这两项功能情况下，程序中调用到的头文件(如#include "DrvI2C.h")，头文件里没用到的函数同样会被编译占用空间。

-funroll-loops 可以添加到 other optimization flags 如图 2，作用是仅对循环次数能够在编译时或运行时确定的循环进行展开，生成的代码尺寸将变大，对于不同的代码执行速度可能变快也可能变慢。

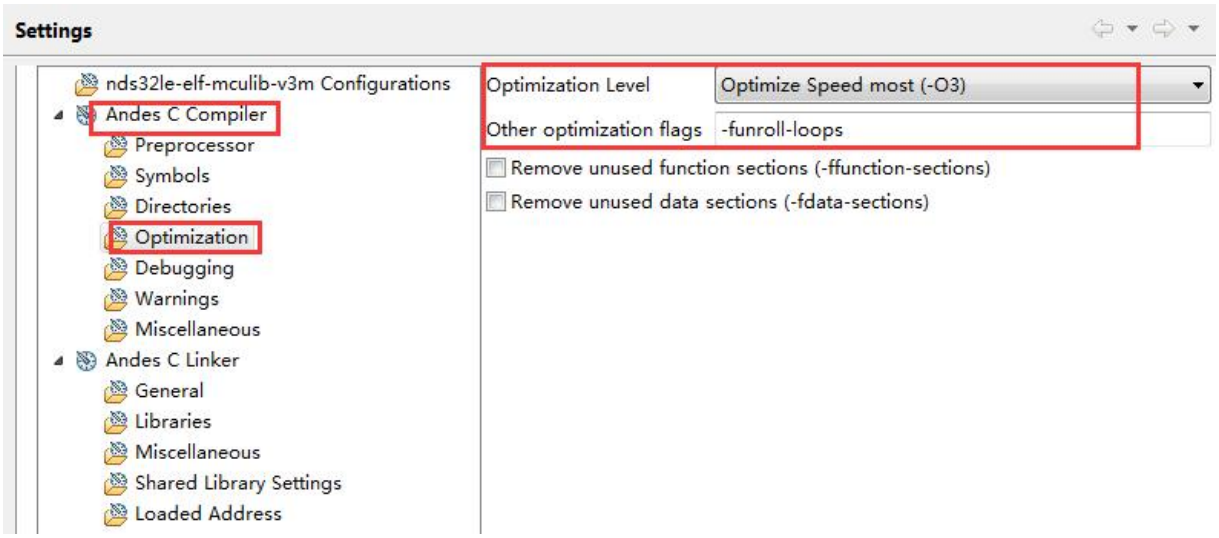


图 2

各优化级别之间更详细的功能介绍可参考：

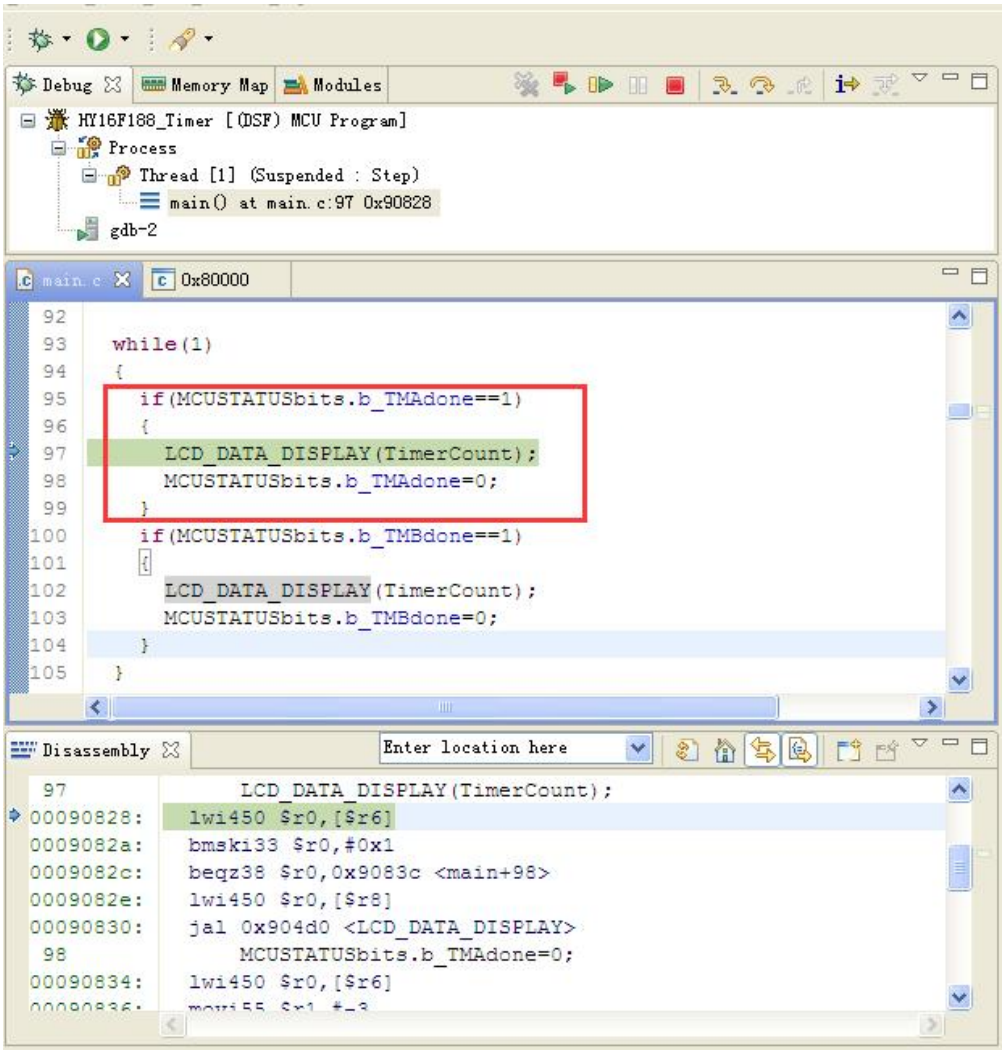
https://gcc.gnu.org/onlinedocs/gcc-4.4.6/gcc/Optimize-Options.html#Optimize-Options

Andes_Programming_Guide 文档

3 使用优化设置需要注意的事项

3.1 调试的问题

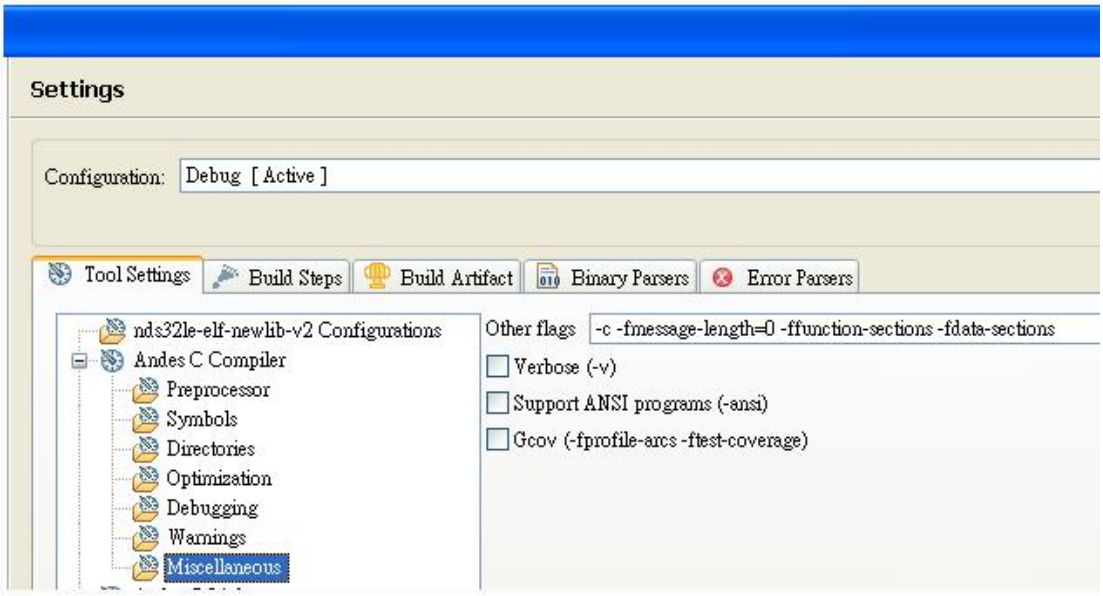
使用以上的优化功能时，由于程序码被优化，所以在 debug 时需要注意可能会遇到 debug 的指标没有对齐到正确的位置，如下图红色框部分，由于事先 MCUSTATUSbits.b_TMAdone 的值已经预设为 0，所以红色框部分的程序属于冗余会被优化掉，虽然 debug 的指标对齐到红色框部分，但是里面的指令可以通过参考下图的 dissassembly 汇编指令的执行情况得知没有被执行，即调试运行的结果是正确的。



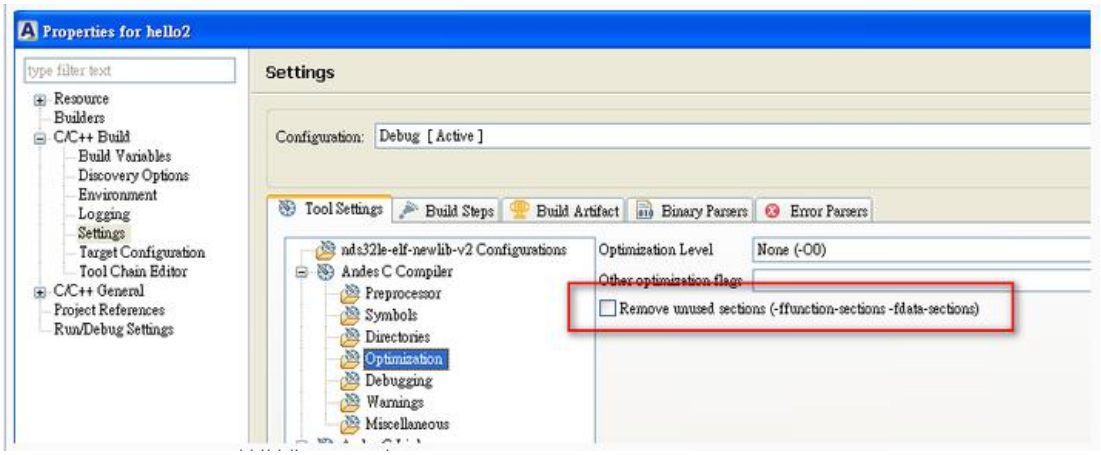
3.2 查询移去的 sections

在-ffunction-section enable 时，要注意是否有用的 section 被 optimize 移除，通常是像 if/else 的 condition 可以在 compiling time 推算出来，所以被 gcc 当成 dead code 优化掉，查询移去哪些 sections 的方法：

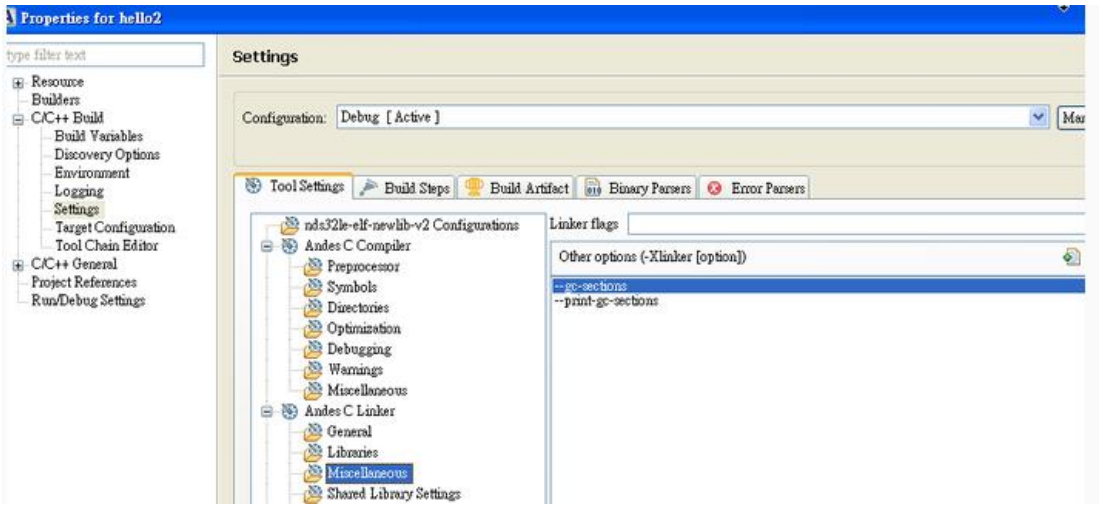
1).在 Andes C Compiler/Miscellaneous 里加上 -ffunction-sections -fdata-sections



2). 取消勾选 Remove unused sections (因为把原本的选项拆开来设定 Remove unused sections 即为 -ffunction-sections -fdata-sections 及--gc-sections)



3). 在 Andes C Linker/Muscellaneous 里加上这 2 个 option :--gc-sections, --print-gc-sections (按下绿色的十字 button 可增加)



4). 我的原始码

```
#include <stdio.h>
#include <stdlib.h>

int int1=34;
int int2=56;
void foo(void);
void foo2(void);

int main(void)
  {
  puts("!!!Hello World!!!"); /* prints !!!Hello World!!! */
  return EXIT_SUCCESS;
}

void foo()
{
  puts("foo"); /* prints !!!Hello World!!! */
}

void foo2()
{
  puts("foo2"); /* prints !!!Hello World!!! */
}
```

5).Build code,原本的 int1 int2 foo1 foo2 都被删掉了如下

```
Terminal | Target Manager : Local Targets | Console
CDT Build Console [hello2]
nds32le-elf-gcc -static -Xlinker --gc-sections -Xlinker --print-gc-sections -O0 -
e:/andestech/andesight200mcu/toolchains/nds32le-elf-newlib-v2/bin/../lib/gcc/nds3
e: Removing unused section '.data.int1' in file './src/hello2.o'
e:/andestech/andesight200mcu/toolchains/nds32le-elf-newlib-v2/bin/../lib/gcc/nds3
e: Removing unused section '.data.int2' in file './src/hello2.o'
e:/andestech/andesight200mcu/toolchains/nds32le-elf-newlib-v2/bin/../lib/gcc/nds3
e: Removing unused section '.text.foo' in file './src/hello2.o'
e:/andestech/andesight200mcu/toolchains/nds32le-elf-newlib-v2/bin/../lib/gcc/nds3
e: Removing unused section '.text.foo2' in file './src/hello2.o'
Finished building target: hello2.adx
```

6).在 project 出现红色 x 不用理会它。只是因为有输出 message。它还是能正常产生结果。有.adx 档输出。

3.3 避免某些程序码被 optimize 的方法

参考语法：http://gcc.gnu.org/onlinedocs/gcc/Funct ... agmas.html

下面提供 2 个写法，其中写法 2 会有 warning 出现，说这个语法在这个 machine 不支援，其实它的结果是正确的。

备注：关于设置单个函数的优化级别 optimize ("Ox")是不支持"Os1" and "Os2"，因为"Os1" and "Os2" 是由 Andes 定义的，不是标准的 GCC 优化级别

写法 1: 只针对 1 个 function（即 add()）。

```
__attribute__((optimize("O0")))   //设置优化级别属性为不优化
int add (int a, int b )
{
int x = a;
int y = b;
return x + y;
}

int main ()
{
int r = 1;
int a = r;
int b = r;
func ();
return 0;
}
```

写法 2：包含在里面的 code 都不会被 optimize。

注意！这个写法不一定要以 function 为范围，

可以任意选取一段 code。

```
#pragma GCC push_options
#pragma GCC optimize ("O0")          //优化级别为"O0"
int add (int a, int b )
{
int x = a;
int y = b;
return x + y;
}
#pragma GCC pop_options

int main ()
```

```
{
int r = 1;
int a = r;
int b = r;
func ();
return 0;
}
```

备注：#pragma GCC push_options

#pragma GCC pop_options

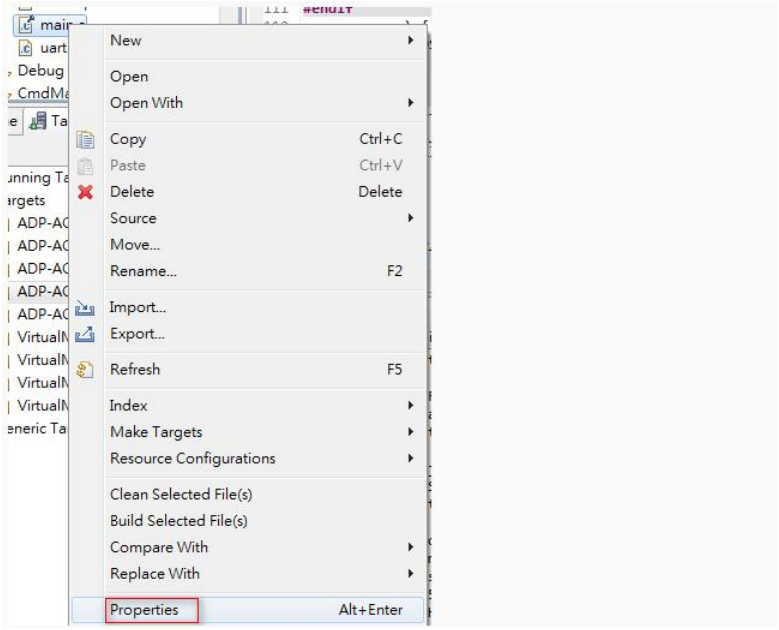
这 2 行是表示会先把原来的 option push 进去，

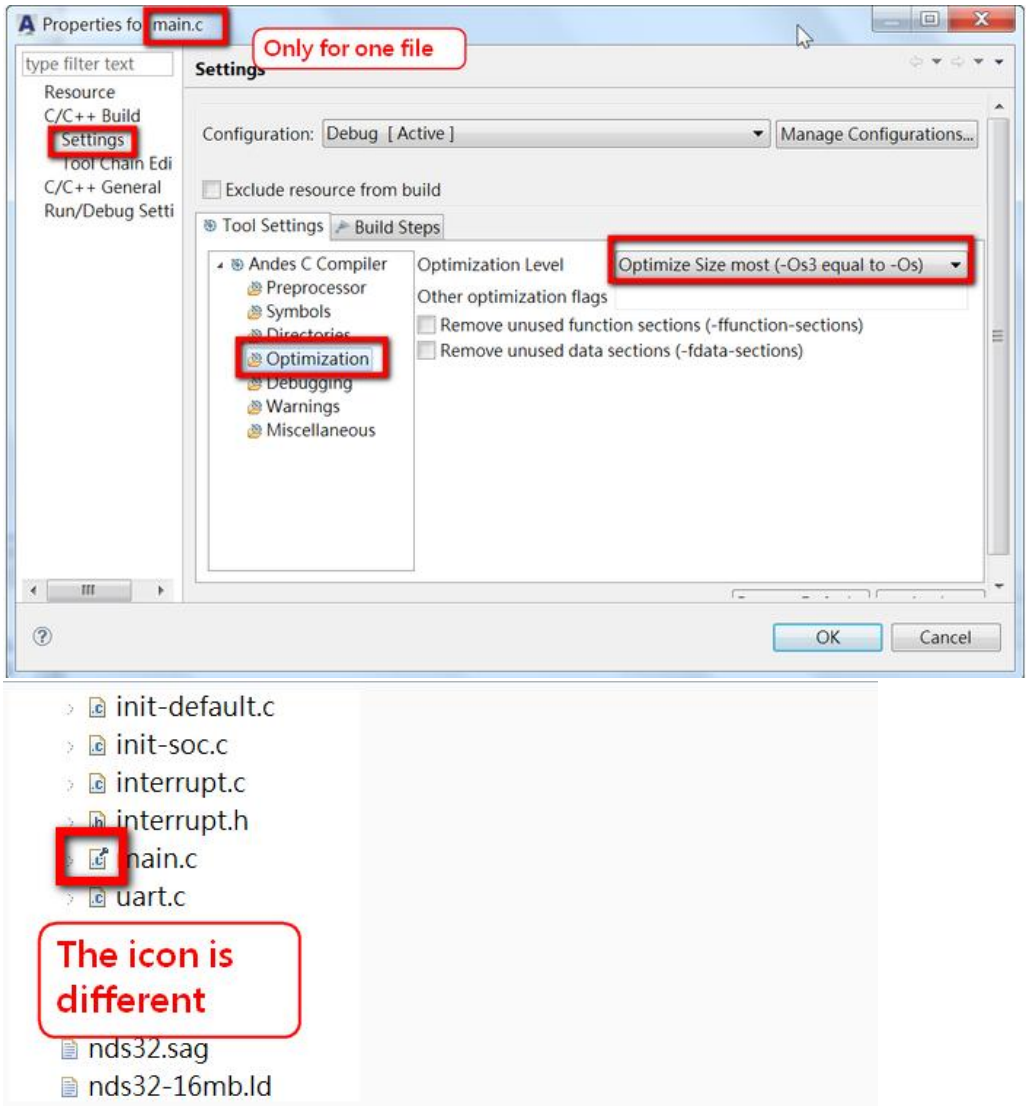
例如是-Os，后面再把-Os pop 回来，恢复原来的 optimize 设定。

3.4 部分代码 optimize 的方法

3.4.1 单个文件设置优化：

鼠标右击源代码文件后，参照下图进行优化级别设置





3.4.2 部分 code 设置优化级别

参考语法:

http://gcc.gnu.org/onlinedocs/gcc/Funct ... agmas.html

代码如下：

```
#pragma GCC push_options
#pragma GCC optimize ("O0")          //优化级别为"O0"，可以设置为其他级别除了 Os1 和 Os2

// code ...                          //程序码
#pragma GCC pop_options
```

备注：#pragma GCC push_options

#pragma GCC pop_options

这 2 行是表示会先把原来的 option push 进去，

例如是-Os，后面再把-Os pop 回来，恢复原来的 optimize 设定。

3.4.3 单个函数设置优化级别

对单个函数的优化级别进行设置，以下代码是设置成’’Os’’

```
void __attribute__ ((optimize ("Os"))) __cpu_init()
{
unsigned int tmp;
/* turn on BTB */
tmp = 0x0;
__nds32__mtsr(tmp, NDS32_SR_MISC_CTL);

/* Set PSW INTL to 0 */
tmp = __nds32__mfsr(NDS32_SR_PSW);
tmp = tmp & 0xfffffff9;
/* ....*/
return;
}
```

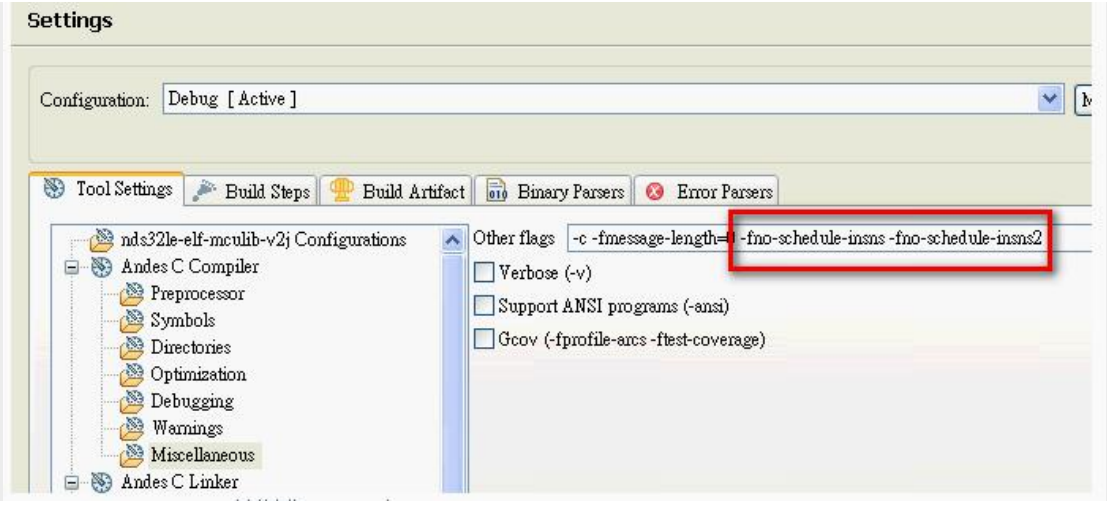
关于设置单个函数的优化级别 optimize ("Ox")是不支持"Os1" and "Os2"，因为"Os1" and "Os2"是由

Andes 定义的，不是标准的 GCC 优化级别

3.5 Optimization 时，避免 code 顺序被改掉的方法

如果不要让 optimization 改变程序码的顺序，

可以加 2 个 options：-fno-schedule -insns -fno-schedule-insns2



3.5 其他需要注意的事项

(1)和硬件相关的变量与 Register 要加上 volatile.加上 volatile 这个关键字避免变量被最佳化 ,例如以下程序：

```
XBYTE[2]=0x55;
XBYTE[2]=0x56;
XBYTE[2]=0x57;
XBYTE[2]=0x58;
```

编译器优化功能会认为只有 XBYTE[2]=0x58(即忽略前三条语句 ,合并为一条语句)。如果键入 volatile，则编译器会逐一的进行编译并产生四条代码相应的机器代码。

(2)在 c code 宣告变量有加 volatile，在 extern 时变量也是要加上 volatile

(3)中断服务程序中修改供其他函数使用的变量建议加上 volatile，因为开启优化功能后，对变量的访问可能借用暂存器，而不是直接访问变量地址，结果可能导致变量地址的值改变没有反映到暂存器上面

(4)多线程应用中被几个任务共享的变量也建议加上 volatile

4 参考文件

Andes_Programming_Guide_v1.5_PG009_V2.1

部分内容摘自 http://forum.andestech.com

5 修订记录

以下描述本档差异较大的地方，而标点符号与字形的改变不在此描述范围。

版本	页次	变更摘要	日期
V01	All	初版发行	2022/08/31